REMARKS

The comments of the applicant below are each preceded by related comments of the examiner (in small, bold type).

2. The specification is objected to as failing to provide proper antecedent basis for the claimed subject matter. See 37 CFR 1.75(d)(l) and MPEP 5608.01(0). Correction of the following is required: a tangible medium and a tangible medium is not defined.

Amendments have been made.

Response to Arguments

3. Applicant's arguments with respect to claims 1, 52, 57 and 68 have been considered but are moot in view of the new ground(s) of rejection.

Applicant argues "thus, no combination of Hapner and Zaiken would have been disclosed or suggested "an index that identifies a contention space of jobs that have competing requirement to write into the region of the database, the index distinguished the contention space from the other content spaces of jobs that do not have competing requirements to write into the region of the database."

The examiner respectfully disagrees with the above argument. Hapner discloses the concept of multithreads processing with different processors including writing which will lock down a specific region of database for processing. What missing from Hapner that the usage of index identified the location of database to be process.

Zaiken discloses the usage of index which served the same purpose of locating the region of database for process.

4. Claims 1-27, 32-43 and 48-51 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hapner et al. (US. Patent No. 5,727,203) in view of Somani et al. (US. Patent No. 5,524,212) and further in view of Fleischman (US. Patent No. 6,507,847)

Regarding on claim 1, Hapner teaches a method comprising: maintaining a database that stores data persistently (col. 7, lines 15-16), accepting tasks from the task sources, at least some of the tasks having competing requirements (multiple threads competing) for user of regions of the database (resources), each of the regions including data that is all either lock or not locked (commit) for writing at a given time (col. 9, lines 54-65).

Hapner does not explicitly teach assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and distributing the jobs for concurrent execution by the associated processors. Somani teaches assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors "as a user writes an INSIGHT program 272 defining application tasks and the number of processor used for each task.. .Each configuration specification file defines a set of generic processors and the jobs partitioned among them1' (cot. 27, lines 56-62). This teaches associated task with each processor and only one task for that processor. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify Hapner system to include the one task for one processor as taught by Somani to allow one processor to read or write the data to the database to keep the data consistency.

Furthermore, Fleischman also discloses a large number of threads can be executing in simultaneously within the database. The term "simultaneously" is used in the context known to programmers similar to "multithreading" i.e. multiple threads do not execute in perfectly simultaneous manner unless the server has parallel processor (col. 5, lines 8-20) and multiple small "read" and "write" commands to a disk drive.. .(col. 5, lines 33-36). Fleischman disclose the execution of the multiple write threads, and each of the processor is assign to a specify write to a specific portion of the database.

Furthermore, in the write thread no other processor to be able to access to the lock region which is accessed by other processor. Therefore, Fleischman implicitly discloses the concept of assigning the assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and distributing the jobs for concurrent execution by the associated processors. Therefore, it would have been obvious to one ordinary skill in art at the time of the invention was made to modify both Hapner and Somani system to include executing of the multiple write threads, and each of the processor is assigned to a specify write to a specific portion of the database and no other processor be able to access to the lock region which is accessed by other processor in the write thread as disclosed by Fleischman to resolve any conflict of accessing the database.

Claim 1 recites maintaining a database that stores data persistently and accepting tasks from task sources. At least some of the tasks have competing requirements for use of regions of the database. Data included in a given region are not available for simultaneous access for writing by more than one of the tasks having competing requirements and data included in different regions are available for simultaneous access for writing by more than one of the tasks having competing requirements. Each of the regions is assigned to a respective available processor, each of the regions being assignable to any of the processors. For each of the tasks, jobs are defined each of which requires write access to a region that is to be accessed by no more than one of the processors, The jobs are distributed for concurrent execution by the assigned processors.

Hapner neither describes nor would have made obvious the features of claim 1. For example, Hapner does not assign each of the regions of a database to a respective available processor. Neither does Hapner define, for each of the tasks, jobs each of which requires write access to a region that is to be accessed by no more than one of the processors let alone distributing the jobs for concurrent execution by the assigned processors.

Neither Somani, nor Fleischman, alone or together, remedies Hapner's deficiencies.

Somani executes jobs in parallel that are partitioned among the processors. As Somani explains:

> A user writes an INSIGHT program 272 defining application tasks and the number of processors used for each task. Translator 274 compiles the program 272 to generate a configuration specification file 275. Each configuration specification file defines a set of generic processors and the jobs partitioned among them. Each file also defines a task class. System 100 executes multiple instances of a single task class or single instances of multiple task classes. (Somani, col. 27, lines 56-67.)

Therefore, although Somani describes partitioning jobs among processors and executing the jobs, Somani does not describe or would not have made obvious assigning each of the regions of a database to a respective available processor, each of the regions being assignable to any of the processors, as recited by claim 1. In fact, Somani is not concerned with regions of a database at all.

Fleischman handles multiple search requests into a database that includes multiple portions on a server at the same time. As Fleischman explains:

> One of the desirable properties of the present invention is that the database is "thread-hot." This terminology refers to a design which can handle multiple search requests into the database at the same time. Each search request, which preferably comes in the form of a function call to the Add( ) function, is considered a "thread." Therefore, the history database can service multiple calls to the Add( ) function at the same time. (Fleischman, col., 4, line 67 to col. 5, line 7.)
>
> Another aspect of the invention is "parallelism", which is closely related to the concept of being thread-hot. The data structure of the present invention is highly parallel, meaning that a very large number of threads can be executing simultaneously within the database. The term "simultaneously" is used in a context known to programmers similar to "multi-threading" i.e. multiple threads do not execute in a perfectly simultaneous manner unless the server has parallel processors. Instead, the threads normally share the resources of a single computer processor. In the present invention, each of the large number of calls to the Add( ) function will make forward progress in reading the database. (Id., col. 5, lines 8-22.)
>
> The structure of the present invention can be further understood by reference to FIG. 1. The history database of the present invention is generally labeled 10 in

> FIG. 1. History database 10 is a "key-value" database, and not a standard relational database structure. The ID of a news article is the key. Each key has a value which is passed as a parameter to the Add( ) function, which is further described herein. The Add( ) algorithm can be described generally as follows: (1) check the database to learn whether the news article is already on the server; (2) if not, add the article to the database and request a download; (3) if the article is already listed in the database then decline the download. (Id., col. 6, lines 34-45.)
>
> A first portion of the database, shown generally as 11 in FIG. 1, is divided into segments, or "buckets." ... A second portion of the database, generally labeled as 12, is reserved for records. (Id., col. 6, lines 52-53 and 56-59.)
>
> When a newsfeed server (or a Usenet user) makes an inquiry to the history database 10, the particular inquiry is referenced by a key. The key is transformed using a hash function... (Id., col. 7, lines 1-4.)
>
> This is shown by line 23 which extends from hash 1, 22, to the corresponding bucket 14. Each bucket, including bucket 14 in this example, contains a pointer to a particular record in section 12. (Id., col. 7, lines 22-23.)

Accordingly, upon each request from the multiple threads, each thread inquires asynchronously the database using a key and accesses the first and second portions of the database based on the result of the inquiry. Although Fleischman does allow multiple threads to inquiry the database in parallel, Fleischman does not disclose or would not have made obvious assigning each of the regions of a database to a respective available processor, as recited by claim 1. In fact, Fleischman's database portions are not assigned to any particular processors but are accessible to multiple newsfeed severs upon their inquiries.

> **As to claim 52, Hapner discloses apparatus comprising:**
> **A database that stores data persistently (col. 7, lines 15-16);**
> **A job processing mechanism that (1) accepting an arbitrarily large number of tasks asynchronously from an arbitrarily large number of task sources, at least some of the tasks having competing requirements for use of regions of database (multiples threads competing for the resources of both database cache 160 and the persistence database portion 164) (col. 9, lines 54-65), each of the regions including data that is all either lock or not locked for writing at a given time (write will lock the database) (col. 9, lines 54-65).**
>
> **Hapner does not explicitly teaches (2) organizes the regions into non-conflicting contention spaces each associated with different available processor, (3) decomposes each of the tasks into jobs each of which require a write access to regions belonging to no more than one of the contention spaces, and (3) distributes the jobs to the corresponding contention spaces for concurrent execution by the associated processor. However, Somani discloses (2) organizes the regions into non-conflicting contention spaces each associated with different available processor, (3) decomposes each of the tasks into jobs each of which require a write**

access to regions belonging to no more than one of the contention spaces, "as a user writes an INSIGHT program 272 defining application tasks and the number of processor used for each task.. .Each configuration specification file defines a set of generic processors and the jobs partitioned among them" (col. 27, lines 56-62). This teaches a write task is associated with one processor and process lock down a part of the database for writing. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify Hapner system to include the one task for one processor as taught by Somani to allow one processor to read or write the data into a part of the database to keep the data consistency. Furthermore, Fleischman also discloses (4) distributes the jobs to the corresponding contention spaces for concurrent execution by the associated processor. The term "simultaneously" is used in the context known to programmers similar to "multithreading" i.e. multiple threads do not execute in perfectly simultaneous manner unless the server has parallel processor (col. 5, lines 8-20) and multiple small "read" and "write" commands to a disk drive. (col. 5, lines 33-36). Fleischman disclose the execution of the multiple write threads, and each of the processor is assign to a specify write to a specific portion of the database. Furthermore, in the write thread no other processor to be able to access to the lock region which is accessed by other processor. Therefore, Fleischman implicitly discloses the concept of assigning the assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and

distributing the jobs for concurrent execution by the associated processors. Therefore, it would have been obvious to one ordinary skill in art at the time of the invention was made to modify both Hapner and Somani system to include executing of the multiple write threads, and each of the processor is assigned to a specify write to a specific portion of the database and no other processor be able to access to the lock region which is accessed by other processor in the write thread as disclosed by Fleischman to resolve any conflict of accessing the database.

For at least the same reasons given for claim 1, claim 52 is patentable.

5. Claims 57-61 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hapner et al. (US. Patent No. 5,727,203) in view of Zaiken et al. (US. Patent No. 5,907,848).

Regarding on claim 57, Hapner teaches a tangible medium bearing instruction to cause a processor to execute a job a job (threads), the job requiring access (write to the database 159) (col. 9, lines 25-30) to a region of a database that stores data persistently (persistence data portion 164) (col. 9, lines 55-60), the job including instruction and pointers to data in the region of the database (col. 9, lines 55-60);

Hapner teaches "as corresponding to multiples threads competing for the resource of both the database cache 160 and the persistence database portion 164" (col. 9, lines 55-60). Hapner does not explicitly teach the job that is executed being selected from a contention space of jobs identified by an index, the jobs in the contention space having competing requirement to write into the region of the database, the index distinguish the contention space from other contention spaces of jobs that do not have competing requirements to write into the region of the database. On the other hand, Zaiken teaches an index that identifies a contention space of jobs that have competing requirement to write into the region of the database, the index distinguish the contention space from other contention spaces of jobs that do not have competing requirements to write into the region of the database "as corresponding to if the file name in the records 20 matches a filename in the selected template 28, the transaction monitor program then searches for any existing index files 30 having job identifier data equal to the job identifier data in the record 20" (col. I I, lines 39-43). This teaches the index identifying the job in the index files to distinguish by comparing the job identifier in the file. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify Hapner system to include comparing the job

identifier in the index files of Zaiken on order to distinguish the jobs by comparing the job identifier in order to process the requested job.

Claim 57 recites causing a processor to execute a required access to a region of a database that stores data persistently. The job includes instructions and pointers to data in the region of the database and is selected from a contention space of jobs identified by an index. The jobs in the contention space have competing requirements to write into the region of the database. The index distinguishes the contention space from other contention spaces of jobs that do not have competing requirements to write into the region of the database.

Hapner did not describe and would not have made obvious the features of claim 57. For example, Hapner does not disclose a job that includes instructions and pointers to data in the region of the database and is selected from a contention space of jobs identified by an index, where the jobs in the contention space have competing requirements to write into the region of the database and the index distinguishes the contention space from other contention spaces of jobs that do not have competing requirements to write into the region of the database.

Zaiken does not remedy Hapner's deficiencies.

Zaiken identifies files affected during a transaction in a database using filenames. In particular, Zaiken creates templates that define transactions and use templates to determine whether each record or entry in the database represent part of a transaction. As Zaiken explains:

> A method and system of providing external transaction protection for a database using the database log or journal. The method involves creating a set of transaction templates which define transactions, using the templates to determine whether each record or entry in the journal represent part of a transaction, and maintaining a set of index file indicating transactions in progress. Each transaction template contains a number of filenames identifying files in the database affected during the transaction defined by the template. Each template also contains a key value representing one or more data fields in the database included in every action performed during the transaction. (Zaiken, Abstract.)

> Once a transaction has started, no other action should be performed on the files involved in the transaction until the transaction is completed or aborted, or else the database could become inconsistent. START, COMMIT, and ABORT statements can be used to determine when and which files are locked. For example, in the log section shown in FIG. 1, filename F1 would be locked until transaction 1 is committed or aborted. (Col. 2, lines 57-65.)

> If the filename in the record 20 matches a filename in the selected template 28, the transaction monitor program then searches for any existing index files 30 having job identifier data equal to the job identifier data in the record 20, step 112. (Col. 11, lines 39-43.)
>
> The key value in the template is used, when the transaction monitor program is operating in a MONITOR mode, to determine whether each log record 20 belongs to a transaction and to select one of the templates 28 based on data contained in the log record 20. (Col. 6, lines 25-30.)

Therefore, each record contains filenames associated with a transaction. However, different transactions can perform action on the same file sequentially, and thus different records or templates associated with different transactions can include the same filenames. Accordingly, the "index" of Zaiken does not identify contention spaces that are distinguished from other contention spaces of jobs that do not have competing requirements to write into the region of the database.

Regarding on claim 68, Hapner teaches a method comprising:

Maintaining a database that stores data persistently (col. 7, lines 15-1 6) and provides a primary level of guarantee that data written in a request transaction is not lost once the transaction is committed (col. 10, lines 11-22),

Accepting tasks from the task source for concurrent execution by multiple processors, at least some of the tasks having conflicting requirements to write into the same region of the database (col. 9, lines 54-65), and

Hapner teaches, "a mutex is created to corresponding to a piece of code, a portion data, some state, etc ... when a thread has locked a mutex, it is said to "own" the locked mutex. In order for other threads to own the mutex, the first thread (i.e. the thread that locked the mutex) must unlock it. Thus mutexes provide a mechanism by which the programmer can control the serialization of multiple threads, ensuring that steps occur in a desired order and that the state corresponding to the mutex is maintained in a consistent manner" (cot. 10, lines 11-22). This teaches the mutex lock is the guarantee that the thread holding the job will be executed and no data is lost.

Hapner does not explicitly teaches providing a software mechanism that guarantees, as least to the primary level of guarantee, that the tasks will be executed without loss of data and without occurrence of any actual conflict with respect to the region of the database. Further more, Fleischman discloses providing a software mechanism that guarantees, as least to the primary level of guarantee, that the tasks will be executed without loss of data and without occurrence of any actual conflict with respect to the region of the database (a large number of threads can be executing in simultaneously within the database. The term "simultaneously" is used in the context known to programmers similar to "multithreading" i.e. multiple threads do not execute in perfectly simultaneous manner unless the server has parallel processor (col. 5, lines 8-20) and multiple small "read" and "write" commands to a disk drive.. .(col. 5, lines 33-36). Fleischman disclose the execution of the multiple write threads, and each of the processor is assign to a specify write to a specific portion of the database. Furthermore, in the write thread no other processor to be able to access to the lock

region which is accessed by other processor. Therefore, Fleischman implicitly discloses the concept of assigning the assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and distributing the jobs for concurrent execution by the associated processors These multiple write thread execution in concurrently will guarantee that tasks will be executed without the loss of data and without occurrence of any actual conflict with respect to the region of the database. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify Hapner system to include multiple write threads concurrently execution with different portions of data as taught by Fleischman in order guarantee that all tasks will be executed without conflict and loss of data.

Claim 68 recites maintaining a database that stores data persistently and providing a primary level of guarantee that data written in a requested transaction is not lost once the transaction is committed. Tasks are accepted from task sources for concurrent execution by multiple processors. At least some of the tasks have conflicting requirements to write into the same region of the database. A software mechanism guarantees, as least to the primary level of guarantee, that the tasks will be executed, and will be executed without loss of data and without the occurrence of any actual conflict with respect to the region of the database.

Hapner does not describe or would not have made obvious the features of claim 68. For example, Hapner does not accept tasks from the task source for concurrent execution by multiple processors, at least some of the tasks having conflicting requirements to write into the same region of the database, and provide a software mechanism that guarantees, as least to the primary level of guarantee, that the tasks will be executed, and will be executed without loss of data and without the occurrence of any actual conflict with respect to the region of the database.

Fleischman does not remedy Hapner's deficiencies. As explained above, Fleischman's multiple threads inquire of different portions, for example, the keys, the first and second portions, asynchronously.

All of the dependent claims are patentable for at least similar reasons as those for the claims on which they depend are patentable.

Canceled claims, if any, have been canceled without prejudice or disclaimer.

Any circumstance in which the applicant has (a) addressed certain comments of the examiner does not mean that the applicant concedes other comments of the examiner, (b) made arguments for the patentability of some claims does not mean that there are not other good

reasons for patentability of those claims and other claims, or (c) amended or canceled a claim does not mean that the applicant concedes any of the examiner's positions with respect to that claim or other claims.

The fee in the amount of $230 for the Petition for Extension of Time fee is being paid on the electronic filing system by way of deposit account authorization. Please apply any other charges or credits to deposit account 06-1050, reference 11811-008002.

Respectfully submitted,

Date: 6/10/8

David L. Feigenbaum
Reg. No. 30,378

Fish & Richardson P.C.
225 Franklin Street
Boston, MA 02110
Telephone: (617) 542-5070
Facsimile: (617) 542-8906

21840463.doc